Illinois Institute of Technology Department of Computer Science

Final Exam (Cumulative)

CS 440— Programming Languages
Fall 2006
December 11, 2006 14:00–16:00

This is a **closed book** and **closed notes** exam.
You are **not** allowed to use calculators or computers during this exam.
Do **ALL** problems in this booklet. Read each question very carefully.
You may detach pages, but **you must return all pages of this exam.**

Name	
Email ID	@iit.edu

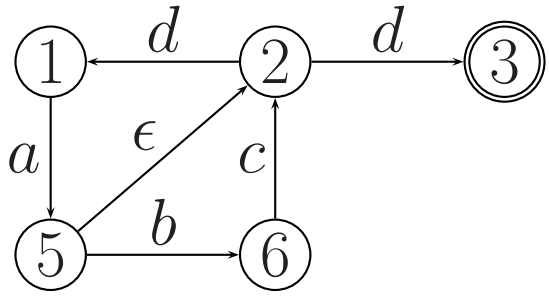
Do **not** place your social security number anywhere on this exam.

Problem	Points	Score
1	6	
2	6	
3	6	
4	6	
5	4	
6	4	
7	6	
8	6	
9	6	
10	6	
11	6	
12	6	
13	4	
14	6	
15	6	
16	6	
17	6	
18	6	
19	6	
20	3	
21	3	
22	3	
23	3	
24	6	
25	4	
Total	130	
Percent	100	

1 Automata

Question 1) (6 points) What is the purpose of an NFA or DFA in the context of programming languages?

Question 2) (6 points) Convert the following NFA to a DFA. (State 1 is the start state.)



2 Grammars

Question 3) (6 points) What does it mean for a grammar to be *ambiguous*? Give an example of an ambiguous grammar, and prove its ambiguity.

Question 4) (6 points) What is the difference between LL and LR grammars? For both of them, give an advantage over the other.

3 Higher Order Functions

Question 5) (4 points) Write the code for `map : ('a -> 'b) -> 'a list -> 'b list`.

Question 6) (4 points) Write the code for `fold_right: ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`.

Question 7) (6 points) Using either `map` or `fold_right`, write the function `product xx : int list -> int`, which computes $\Pi_{i=0}^{n} a_i$ from the list $[a_0; a_1; \ldots; a_n]$. You may **not** use explicit recursion.

```
# let product xx = ...
val product : int list -> int = <fun>
# product [3;1;9];;
- : int = 27
```

Question 8) (6 points) Using either `map` or `fold_right`, write the function `decList xx : int list -> int list`, which decrements each element of a list. You may **not** use explicit recursion.

```
# let decList xx = ...
val decList : int list -> int = <fun>
# decList [2;4;6];;
- : int list = [1;3;5]
```

4 Parameters

Question 9) (6 points) Consider the following code:

```
let i = 10;;
let f a =
   a := 20;
   a := a * i;
   i;;
print f(i);;
print i;;
```

What will be printed to the screen for the following parameter passing styles?

Call By Value

Call By Reference

Call By (Copy-)Result

Question 10) (6 points)
Consider the following code:

```
let f a b =
   a + a + a + b;;

f (10 * 20) (20 * 30);;
```

1. How many times is * called in the call by name parameter passing style?

2. How many times is * called in the call by need parameter passing style?

Question 11) (6 points) In a call stack, what is the difference between the static link and the dynamic link? Give an example of a language that would need both of these, **and** another example of a language that would **not** need both of these.

5 Prolog

Question 12) (6 points)
Consider the following Prolog code.

```
length([],0).
length([H|T],X) :- length(T,Y), X is Y + 1.
```

Show how to write the `prod` predicate, which is true when the second argument is the product of the elements of the first argument.

Question 13) (4 points) For what kind of applications is Prolog well-suited? Give an example.

Question 14) (6 points) Consider the following Prolog database.

```
connected(a,b).
connected(b,c).
connected(a,d).
connected(d,c).
etc...
```

This database represents the connections in a graph. Write a predicate `pathfrom(X,Y)` which is true when there exists a path from `X` to `Y` in the database. (Assume that other things may be added to the database.)

Question 15) (6 points) Consider the following Prolog database.

```
pet(puppy).
pet(tarantula).
adjective(happy).
adjective(hairy).
adjective(dangerous).
```

If you give the query

```
?- adjective(X), pet(Y).
```

how many results are returned?

Show how to insert a cut operator so that only one result will be returned.

What will that result be?

6 Recursion

Use recursion to write the following function. You do **not** have to use tail recursion, but you may do so if you want. You are **always** allowed to make helper functions.

Question 16) (6 points) Write a recursive function `sumonen : int -> int` that takes an integer n and returns the sum $\Sigma_{i=1}^{n} i$. If $n < 0$ simply return 0.

```
# let rec sumonen n = ...
val sumonen : int -> int = <fun>
# sumonen 10;;
val - : int = 55
```

Question 17) (6 points) Write the function `sum xx : int list -> int`, which returns the sum $\Sigma_{i=1}^{n} a_i$ from the list $[a_0; a_1; \ldots; a_n]$. Use tail recursion. You may write a helper function, or use standard library functions if you want.

7 State

Question 18) (6 points) Suppose you want to write an accumulator function. It keeps a local state, starting at 0, and then keeps a running total of the input. Here is a session:

```
# let acc = some secret stuff
val acc : int -> int = <fun>
# acc 2;;
- : int = 2
# acc 3;;
- : int = 5
# acc 5;;
- : int = 10
```

Give the code for `acc` that has the behavior above. You may not use a global variable.

Question 19) (6 points) State makes it difficult to use equational reasoning. Give an example, using the `acc` function above, that illustrates this.

8 Types

Question 20) (3 points) What is the type of the following function?

```
let foo x = x + x
```

Question 21) (3 points) What is the type of the following function?

```
let bar y = y @ [3]
```

Question 22) (3 points) What is the type of the following function?

```
let baz z = 10 + (z 5)
```

Question 23) (3 points) What is the type of the following function?

```
let gop w = w (w 2.4)
```

9 Unification

Question 24) (6 points) Solve the following unification problem. Show your work to allow for partial credit if something goes wrong. The Greek letters are the variables.

$$\{g(\alpha, x) = g(y, \beta), \quad h(\gamma, z) = h(f(\alpha), z)\}$$

Question 25) (4 points) Give an example of a language feature (from any language) which is implemented using unification.